**Chapter 1 ウィンドウの作成**

DirectX11を使用してゲームを作るためには、まずウィンドウを作成する必要があります。このチャプターではウィンドウを作成する方法を学んでいきましょう。使用するサンプルプログラムはLesson\_00です。

**1.1 ウィンドウクラス**

ウィンドウを作成するためにはウィンドウクラスを登録する必要があります。(注意:

ここでいうクラスはC++のクラスとは一切関係がないので注意してください。単にマイクロソフトがウィンドウクラスと呼んでいるだけです。)

ウィンドウクラスの登録はWNDCLASSEXという構造体とRegisterClassExという関数を使用して下記のように記述します。

Lesson\_00/Lesson\_00.cpp(?行目)

|  |
| --- |
| //ウィンドウクラスのパラメータを設定(単なる構造体の変数の初期化です。)  WNDCLASSEX wc =  {  sizeof(WNDCLASSEX), //構造体のサイズ。  CS\_CLASSDC, //ウィンドウのスタイル。  　　　　　　 //ここの指定でスクロールバーをつけたりできるが、  　　　　　　 //ゲームでは不要なのでCS\_CLASSDCでよい。  MsgProc, //メッセージプロシージャ(後述)  0, //0でいい。  0, //0でいい。  GetModuleHandle(NULL), //このクラスのためのウインドウプロシージャがあるインスタンスハンドル。  //何も気にしなくてよい。  NULL, //アイコンのハンドル。アイコンを変えたい場合ここを変更する。  //とりあえずこれでいい。  NULL, //マウスカーソルのハンドル。NULLの場合はデフォルト。  NULL, //ウィンドウの背景色。NULLの場合はデフォルト。  NULL, //メニュー名。NULLでいい。  TEXT("Sample\_00"), //ウィンドウクラスに付ける名前。  NULL  };  //ウィンドウクラスの登録。  RegisterClassEx(&wc); |

これでウィンドウクラスが登録できました。

**1.2 ウィンドウの作成**

では続いて、1.1で作成したウィンドウクラスを使用してウィンドウを作成する方法を見ていきましょう。ウィンドウの作成にはCreateWindowという関数を使用します。

|  |
| --- |
| // ウィンドウの作成。  g\_hWnd = CreateWindow(  TEXT("Sample\_00"), //使用するウィンドウクラスの名前。  //先ほど作成したウィンドウクラスと同じ名前にする。  TEXT("Sample\_00"), //ウィンドウの名前。ウィンドウクラスの名前と別名でもよい。  WS\_OVERLAPPEDWINDOW, //ウィンドウスタイル。ゲームでは基本的にWS\_OVERLAPPEDWINDOWでいい、  0, //ウィンドウの初期X座標。  0, //ウィンドウの初期Y座標。  500, //ウィンドウの幅。  500, //ウィンドウの高さ。  NULL, //親ウィンドウ。ゲームでは基本的にNULLでいい。  NULL, //メニュー。今はNULLでいい。  hInstance, //アプリケーションのインスタンス。  NULL  ); |

CreateWindow関数は作成したウィンドウのハンドルを返してきます(ウィンドウのアドレスを返してくると考えて構いません)。このウィンドウハンドルは後々使用することになるので、グローバル変数のg\_hWndに記録しています。

ウィンドウの作成ができたら、ShowWindow関数を使用してウィンドウを表示します。

|  |
| --- |
| ShowWindow(g\_hWnd, nCmdShow); |

**1.3 メッセージループとメッセージプロシージャ**

ウィンドウアプリケーションのようなGUIアプリはコンソールアプリケーション(CUIアプリ)と異なり、ユーザーが明示的に終了させるまで、永続的にアプリケーションが起動しています。なので、起動したらすぐにアプリケーションが終了するのではなく、ユーザーがアプリケーションを閉じるまで、ずっと実行し続ける必要があります。これを実現するためのものがメッセージループです。サンプルプログラムにはShowWindowを実行した後で、下記のようなループがあります。

|  |
| --- |
| //メッセージ構造体の変数msgを初期化。  MSG msg = { 0 };  while (WM\_QUIT != msg.message) //メッセージループ  {  //ウィンドウからのメッセージを受け取る。  これがメッセージループ  if (PeekMessage(&msg, NULL, 0, 0, PM\_REMOVE))  {  TranslateMessage(&msg);  DispatchMessage(&msg);  }  } |

**1.3.1 メッセージプロシージャ**

　ウィンドウズプログラムではユーザーの操作に対して、様々なアクションを起こす必要があります。このユーザーの操作によるメッセージを受け取っているのが、メッセージループ内のPeekMessage、TranslateMessage、DispatchMessageという関数です。この三つの関数を使用して、ユーザーからのメッセージを受けとると、メッセージプロシージャという特殊な関数が呼び出されます。これが1.1のウィンドウクラスの作成の時に登録した、メッセージプロシージャです。メッセージプロシージャはアプリケーションが独自に作成する必要がある関数で、アプリケーションごとに実装されています。では具体的なコードを見てみましょう。

|  |
| --- |
| //メッセージプロシージャ。  //hWndがメッセージを送ってきたウィンドウのハンドル。  //msgがメッセージの種類。  //wParamとlParamは引数。今は気にしなくてよい。  LRESULT CALLBACK MsgProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)  {  //送られてきたメッセージで処理を分岐させる。  switch (msg)  {  case WM\_DESTROY:  PostQuitMessage(0);  break;  case WM\_LBUTTONDOWN:  //マウスの左ボタンが押された。  MessageBox(hWnd, "マウスの左ボタンが押されたよ。", "お知らせ", MB\_OK);  break;  default:  return DefWindowProc(hWnd, msg, wParam, lParam);  }  return 0;  } |

これがウィンドウクラスを登録する時に指定していたメッセージプロシージャです。引数のmsgにユーザーが行った操作に関するメッセージが記録されています。このサンプルではWM\_LBUTTONDONW、マウスの左ボタンが押されたときのメッセージ、が送られてきたときにMessageBoxという関数を使用して、メッセージを表示しています。

**1.4 実習課題**

　①　ウィンドウのサイズを1280×720に変更しなさい。

　②　ウィンドウの名前をGameDemoに変更しなさい。

　③　マウスの右ボタンが押されたときに「マウスの右ボタンが押されたよ」というメッセージを表示しなさい。